Stateful REST API Fuzzing with RESTler

FuzzCon-Europe 2021

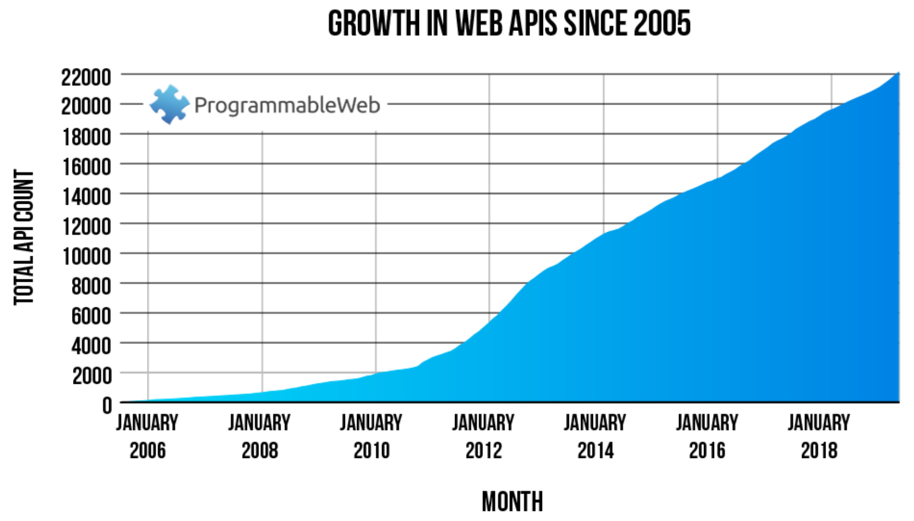
Marina Polishchuk

Microsoft Research

Joint work with: Patrice Godefroid, Vaggelis Atlidakis, Jamie Davis, Richard Files, Bo-Yuan Huang, and Daniel Lehmann

Web APIs Everywhere

- Most cloud services programmatically accessed through REST APIs
- Many cloud backends are microservices with private APIs
- Services are rapidly evolving
  - Need testing that keeps up with the pace of development and deployment
  - How secure and reliable are the APIs?



Outline

- Why Stateful REST API fuzzing?
- Introducing RESTler
- Types of bugs RESTler can find
- CI/CD
- Future work

Why Stateful REST API fuzzing?

Research started in 2017

Motivated by Microsoft-internal user feedback on gaps in existing solutions:

- Fuzzing *each request in isolation*
  - – low coverage – most requests depend on some pre-created resources
  - – human in the loop required to guide the fuzzer
- *Traffic capture* and *replay with fuzzing*
  - – coverage depends on how comprehensive the recorded traffic is
- *Manually written* fuzzers

Why Stateful REST API fuzzing?

Research started in 2017

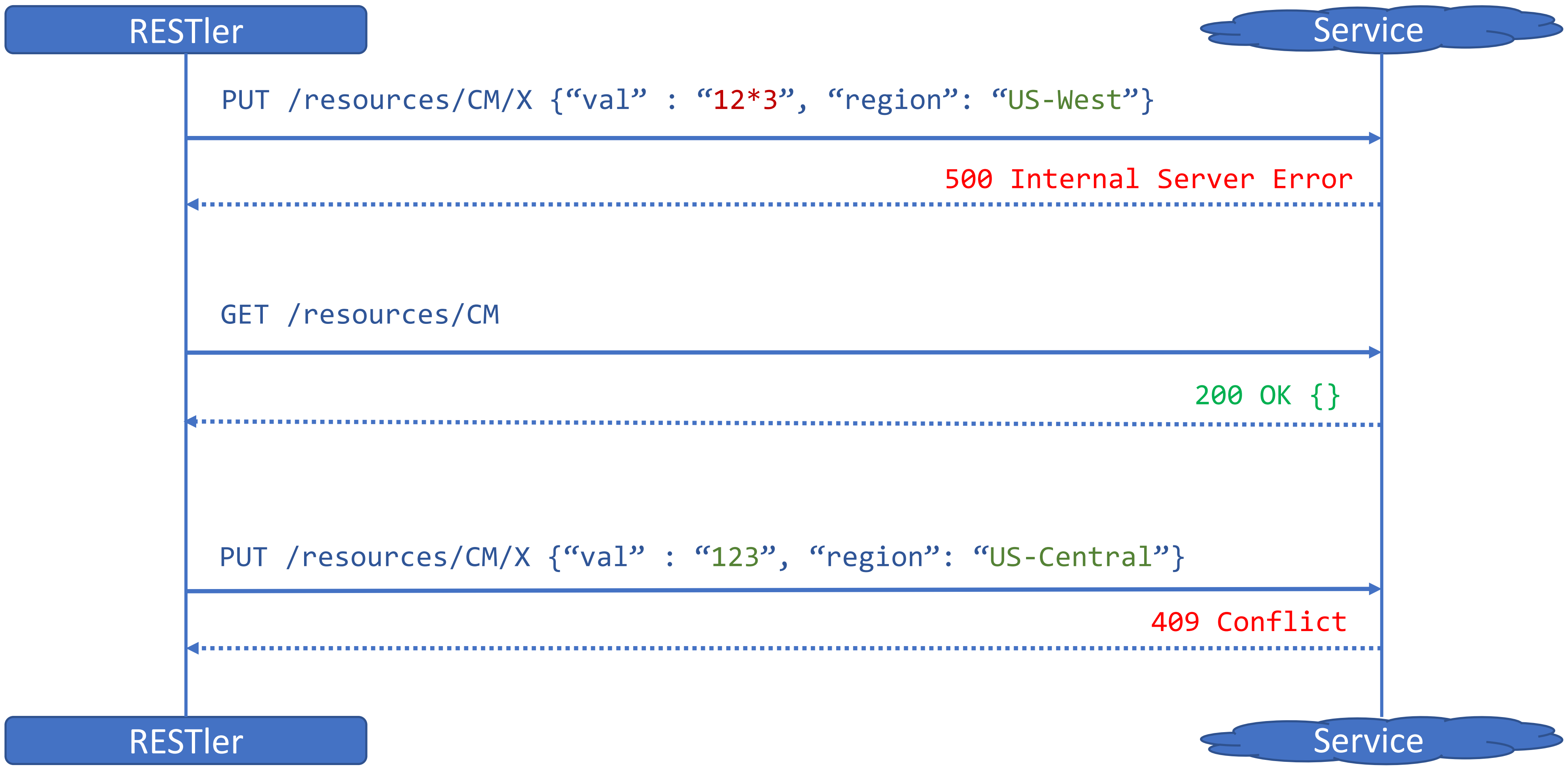
Motivated ng
solutions:

- Fuzzing
  - low urces
  - hur
- *Traffic*
  - coverage depends on how comprehensive the recorded traffic is
- *Manually written* fuzzers

Too expensive to create and maintain for large REST APIs (e.g. Azure services)

Important bugs missed

Example Bug



```
PUT /resources/CM/X {“val” : “12*3”, “region”: “US-West”}
500 Internal Server Error
GET /resources/CM
200 OK {}
PUT /resources/CM/X {“val” : “123”, “region”: “US-Central”}
409 Conflict
```

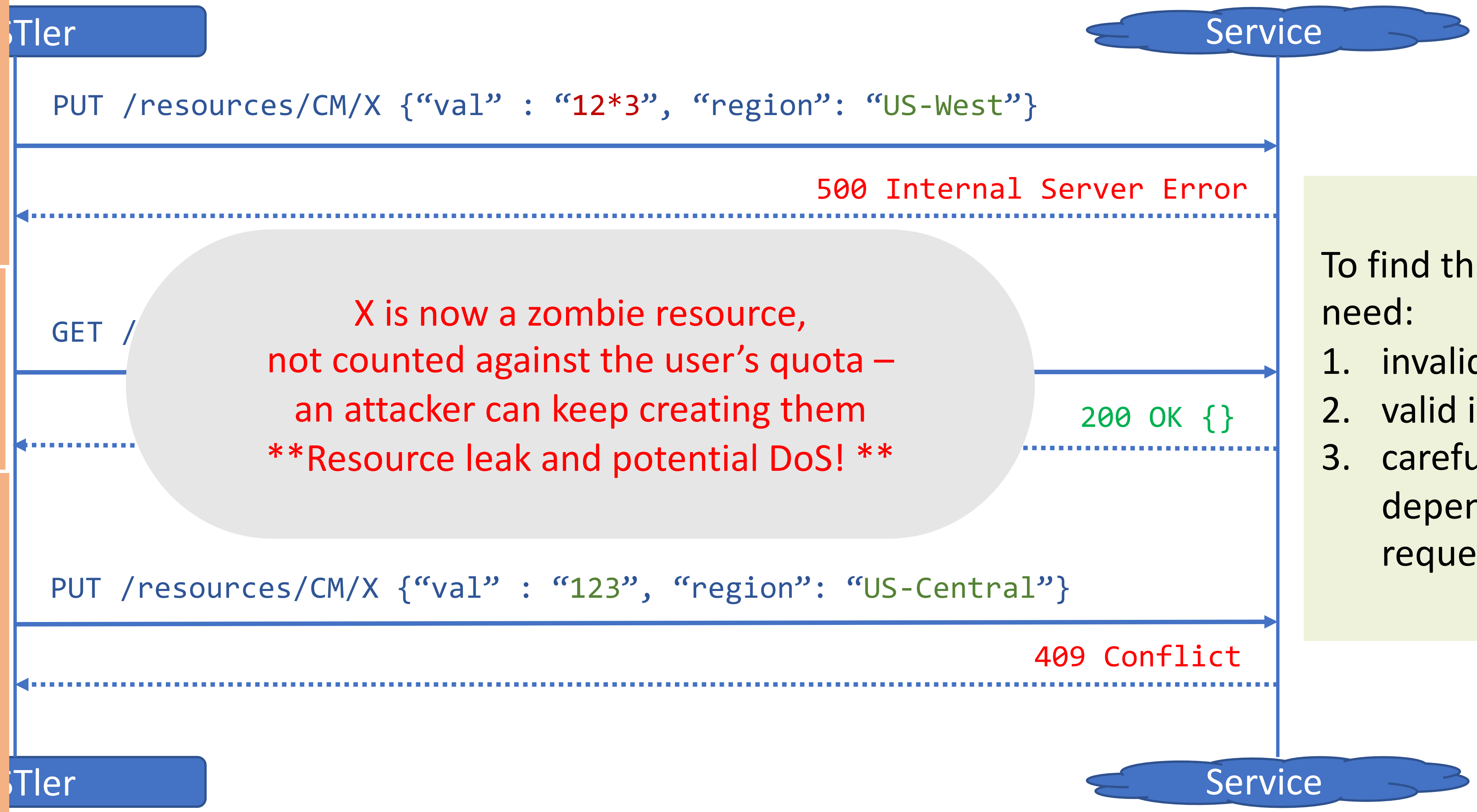
[Checking Security Properties of Cloud Service REST APIs , V. Atlidakis, P. Godefroid, M. Polishchuk, ICST’2020]

Example Bug: Resource Leak

Step 1: create a new resource of type CM with name "X" in *US-West region* with a *malformed body*

Step 2: get a list of all resources of type CM

Step 3: create a new resource of type CM with name "X", but with a *well-formed body* and in a *different region*



STler
Service
PUT /resources/CM/X {"val" : "12*3", "region": "US-West"}
500 Internal Server Error
GET /
200 OK {}
PUT /resources/CM/X {"val" : "123", "region": "US-Central"}
409 Conflict
STler
Service

X is now a zombie resource,
not counted against the user's quota –
an attacker can keep creating them
**Resource leak and potential DoS! **

To find this bug, you need:

1. invalid inputs
2. valid inputs
3. carefully crafted dependent requests

[Checking Security Properties of Cloud Service REST APIs , V. Atlidakis, P. Godefroid, M. Polishchuk, ICST'2020]

RESTler: a Stateful REST API Fuzzer

Stateful:

- Tests are sequences of requests
  - A request is only fuzzed if its pre-requisite resources can be created
- Fuzzing algorithm avoids redundant testing

***Automatic*:**

- Uses the Swagger/OpenAPI specification to generate fuzzing grammar
  - Published for many REST APIs, used to generate client SDKs
- Good coverage out of the box for simple, well-documented APIs

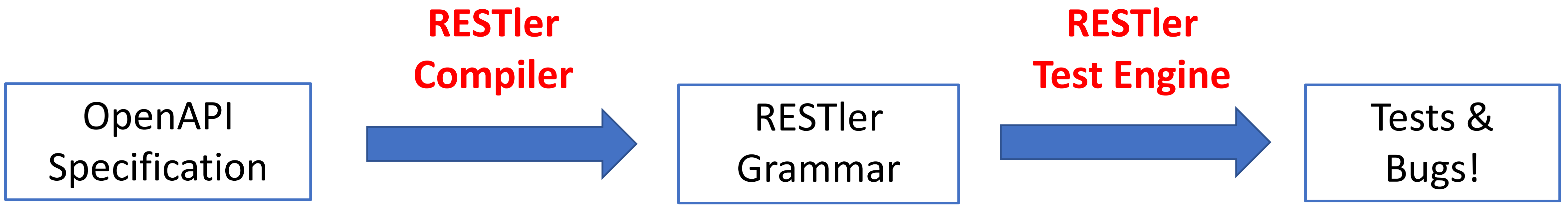
***Extensible*:**

- Pluggable active property checkers

Hundreds of bugs found across Microsoft services in the past 3 years

RESTler open sourced on github November 2020

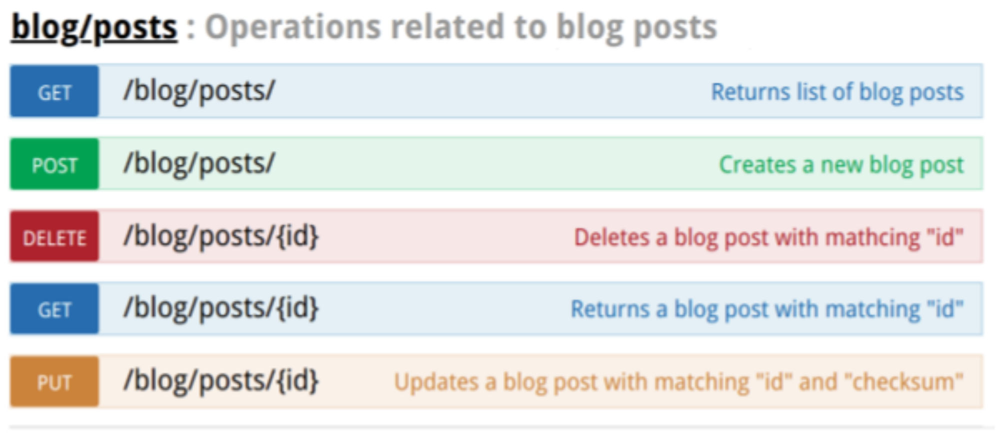
How RESTler works



- Infer how to fuzz each *request type*
- Generate code to parse *responses*
- Identify producer-consumer *dependencies*

- *Generate* & *execute tests*
- *Analyze* test results: feedback loop to *learn* from past service responses
- *Systematic state-space search*

Example



blog/posts : Operations related to blog posts

GET	/blog/posts/	Returns list of blog posts
POST	/blog/posts/	Creates a new blog post
DELETE	/blog/posts/{id}	Deletes a blog post with mathcing "id"
GET	/blog/posts/{id}	Returns a blog post with matching "id"
PUT	/blog/posts/{id}	Updates a blog post with matching "id" and "checksum"

Sample OpenAPI spec
(five requests)

RESTler Compiler

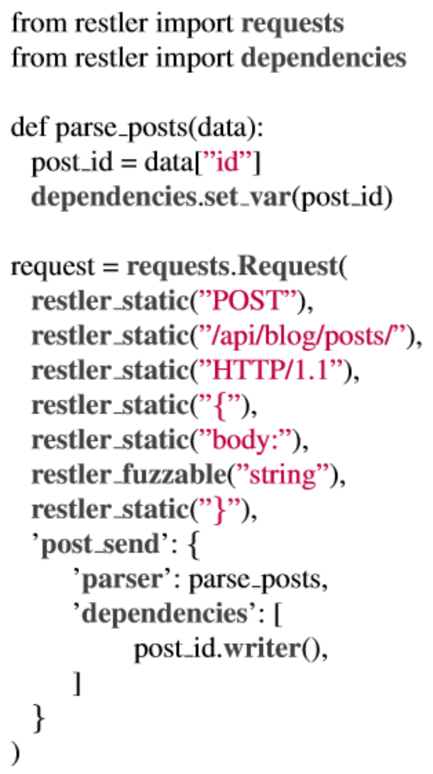
```
from restler import requests
from restler import dependencies

def parse_posts(data):
  post_id = data["id"]
  dependencies.set_var(post_id)

request = requests.Request(
  restler_static("POST"),
  restler_static("/api/blog/posts/"),
  restler_static("HTTP/1.1"),
  restler_static("{"),
  restler_static("body:"),
  restler_fuzzable("string"),
  restler_static("}"),
  'post_send': {
      'parser': parse_posts,
      'dependencies': [
          post_id.writer(),
      ]
  }
)
```

Grammar fragment
(one HTTP request)

Example



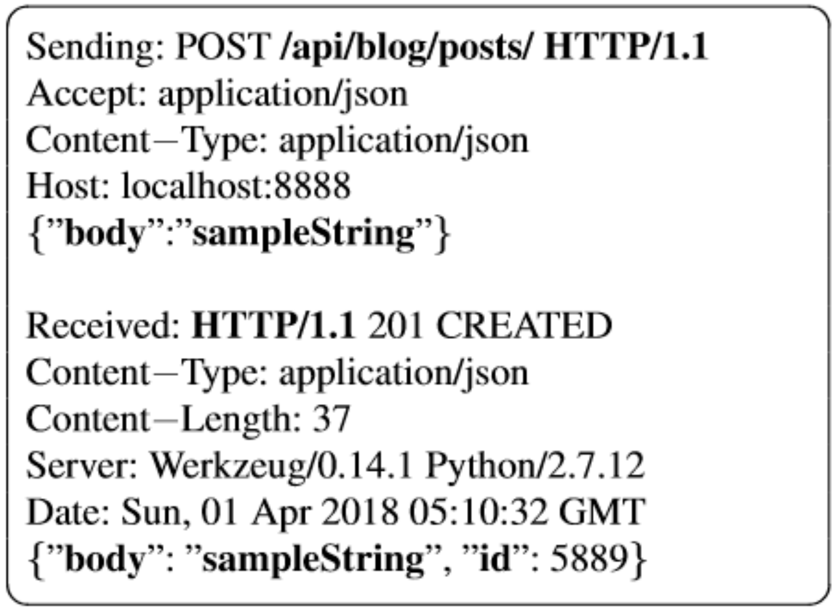
```
from restler import requests
from restler import dependencies

def parse_posts(data):
  post_id = data["id"]
  dependencies.set_var(post_id)

request = requests.Request(
  restler_static("POST"),
  restler_static("/api/blog/posts/"),
  restler_static("HTTP/1.1"),
  restler_static("{"),
  restler_static("body:"),
  restler_fuzzable("string"),
  restler_static("}"),
  'post_send': {
      'parser': parse_posts,
      'dependencies': [
            post_id.writer(),
      ]
  }
)
```

Grammar fragment
(one HTTP request)

RESTler
Test Engine



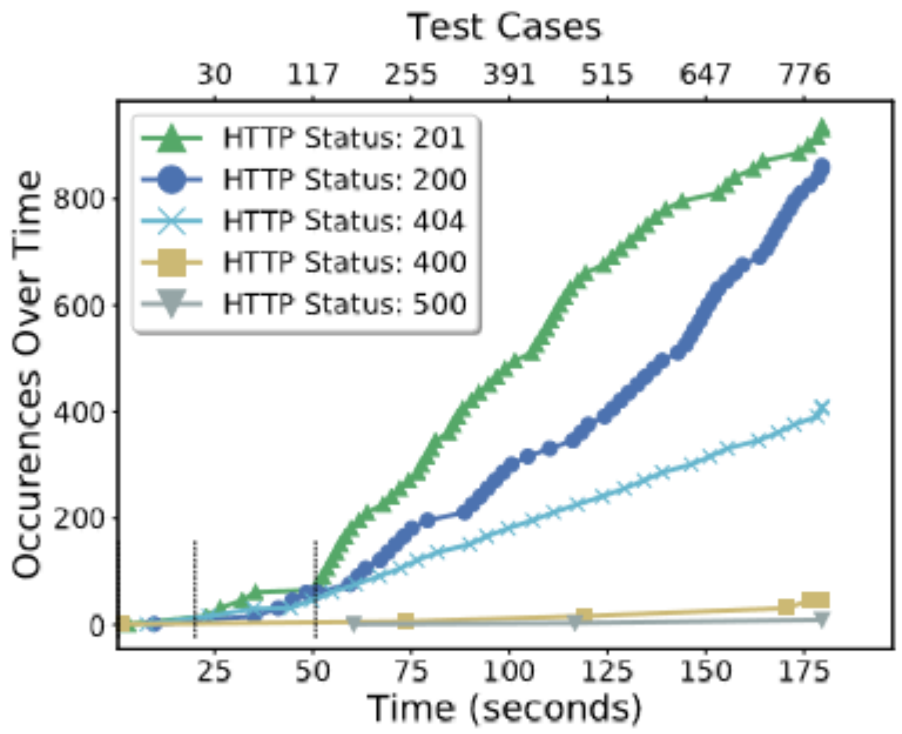
```
Sending: POST /api/blog/posts/ HTTP/1.1
Accept: application/json
Content−Type: application/json
Host: localhost:8888
{"body":"sampleString"}

Received: HTTP/1.1 201 CREATED
Content−Type: application/json
Content−Length: 37
Server: Werkzeug/0.14.1 Python/2.7.12
Date: Sun, 01 Apr 2018 05:10:32 GMT
{"body": "sampleString", "id": 5889}
```

A sample test
(one HTTP request)

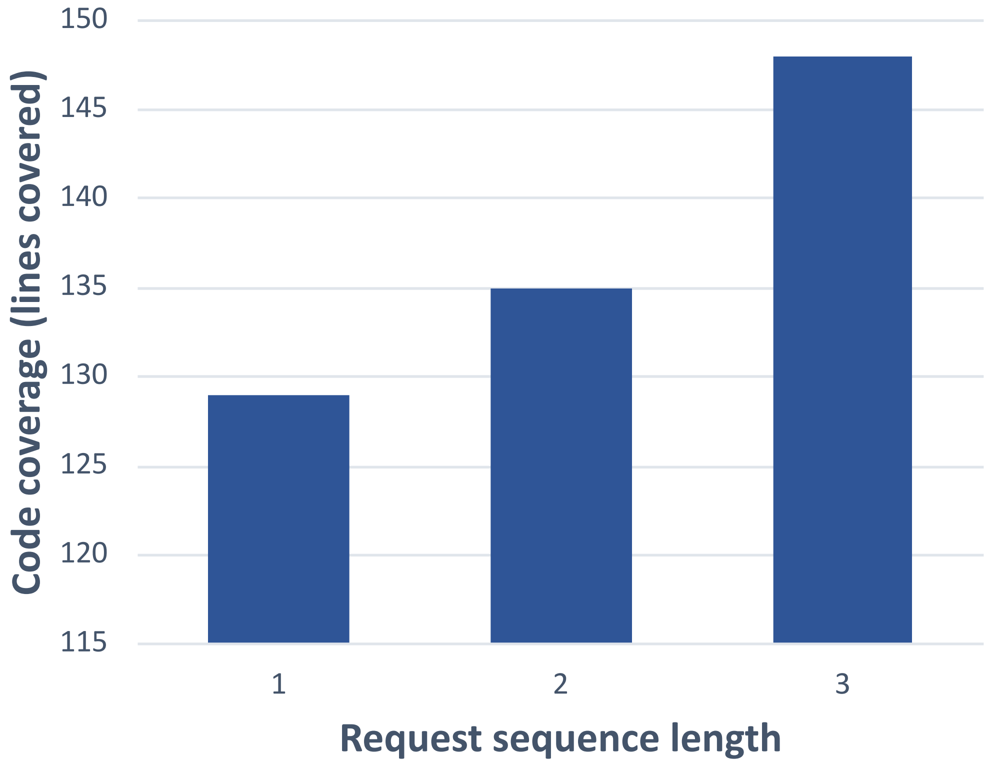
Example

Fuzzing session (100s tests/min)



Example

Fuzzing session (100s tests/min)



Sequence length 1

1. GET /blog/posts
2. POST /blog/posts

Sequence length 2

1. POST /blog/posts { “body”: “a” }
   GET /blog/posts/1
2. POST /blog/posts { “body”: “b” }
   DELETE /blog/posts/2
3. GET /blog/posts
   POST /blog/posts { “body”: 123 }

…

Sequence length 3

1. POST /blog/posts { “body”: “a” }
   PUT /blog/posts/1 {“body”: “”}
   DELETE /blog/posts/1
2. POST /blog/posts { “body”: “a” }
   DELETE /blog/posts/2
   GET /blog/posts

…

Outline

- Why Stateful REST API fuzzing?
- Introducing RESTler
- *Types of bugs RESTler can find*
- CI/CD
- Future work

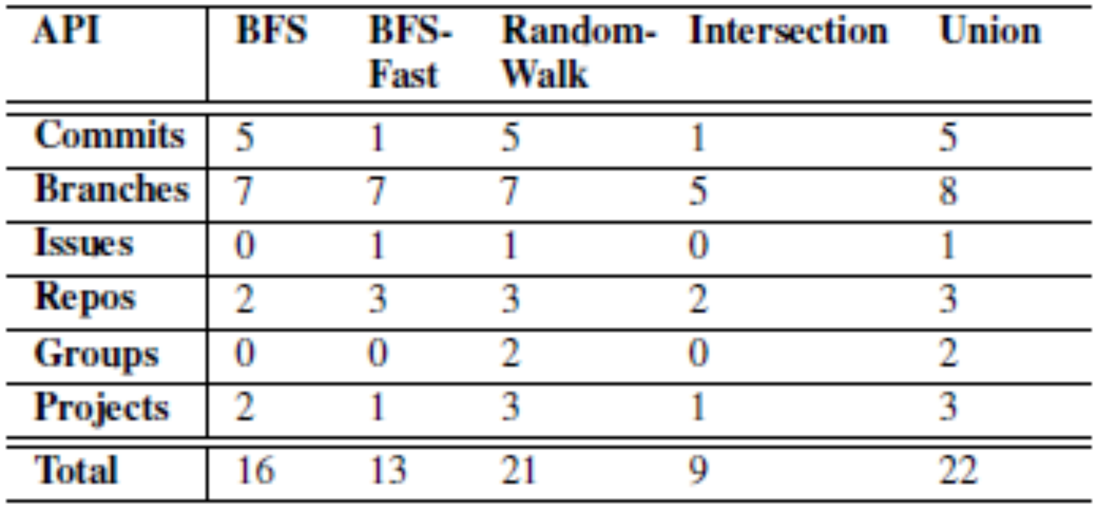
Case Study: GitLab

- Open-source self-hosted Git service (mil of users)
- Complex REST API: ~300 request types
- RESTler found 28 new bugs

Example: 1. Create a project
2. Create a repo file with a proper commit
3. Delete the repo file with an empty commit message
→ "500 Internal Server Error"

All these bugs have been fixed!

(see the [reference] below for details)



API	BFS	BFS-Fast	Random-Walk	Intersection	Union
Commits	5	1	5	1	5
Branches	7	7	7	5	8
Issues	0	1	1	0	1
Repos	2	3	3	2	3
Groups	0	0	2	0	2
Projects	2	1	3	1	3
Total	16	13	21	9	22

TABLE III: **Bug Buckets found by BFS, BFS-Fast, and RandomWalk after Five Hours.** Shows the sets of bugs found by each search strategy in each API. In total: *REST-ler* found 22 new bugs.

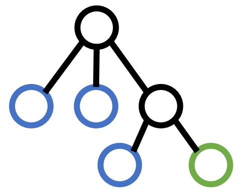
[RESTler: Stateful REST API Fuzzing, ICSE'2019]

Case Study: Azure DNS

- JSON payload fuzzing
  - Fuzz both the schema and data values
  - Dynamically augment set of possible inputs using past responses

```
{
  “location”: {
    “type”: “string”
  },
  “tag”: {
    “type”: “string”
  },
  “properties”: {
    “id”: {
      “type”: “string”,
    },

    “timeout”: {
      “type”: “number”,
    }
  }
}
```



- object
- string
- number

```
{
  “location”: “!@#$%^&”,
  “tag”: “conference-talk”,
  “properties”: {
    “id”: “Microsoft”,
    “timeout”: -1
  }
}
```

DNS service:

- 13/13 covered request types
- 4/13 with non-empty JSON-payload
- Schema size: 2, 22, 65, and 65

- Found **202** different error code/message
- Found new 500 internal server errors
  - In 3 out of 4 request types
  - The one with no bug found has 2 nodes
  - 7 new bugs filed

[Intelligent REST API Data Fuzzing, FSE’2020]

Classes of bugs found by RESTler

- API specification
  - Naming or type hierarchy inconsistencies (for dependencies) *
  - Incorrect examples
- Input validation
  - Unhandled exception (e.g. 500 instead of 400)
- Authentication
  - Unauthenticated APIs *
  - Able to access another user's resources
  - Crashing authentication -> inaccessible service *
- Resource management
  - Resource exhaustion **
  - Inability to create resource after error
  - Create invalid resource (e.g. that can't be referenced/
- Data leaks
  - Leaking debug data types *

All these bugs are being fixed!

Careful when fuzzing in Production

* found during manual investigation

** found by service alerts

When to do REST API fuzzing

- Developers
  - During API development
  - CI/CD regression fuzzing
  - Deployment validation
- Security engineers
  - Bug hunt for specific classes of bugs

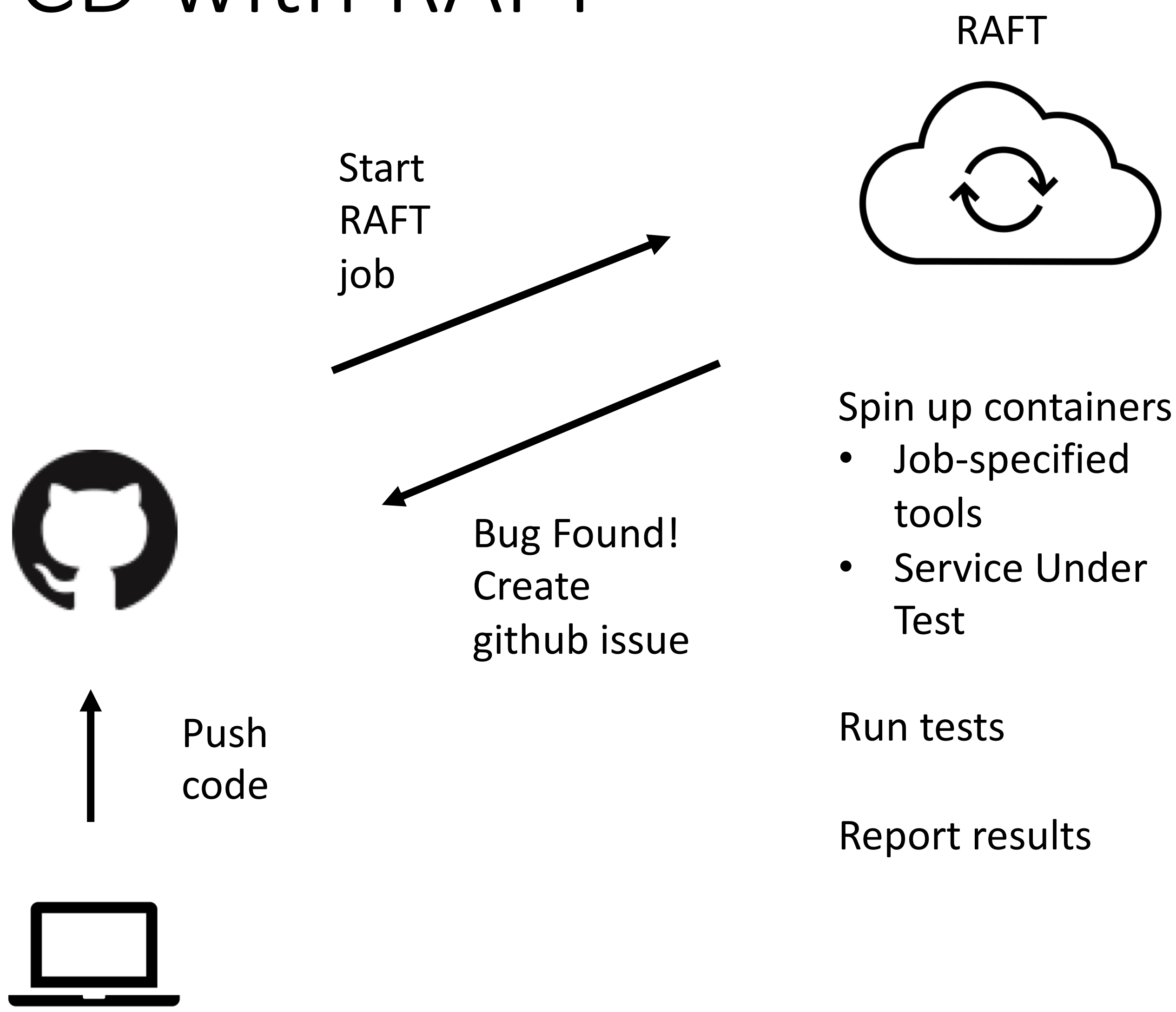
Outline

- Why Stateful REST API fuzzing?
- Introducing RESTler
- Types of bugs RESTler can find
- *CI/CD*
- Future work

RESTler in CI/CD with RAFT

- RAFT is a self hosted REST API Fuzzing-As-A-Service platform
  - Runs on Azure
  - Supports several API fuzzing and scanning tools, easy to onboard new tools
  - Allows deploying service at time of fuzzing, if packaged in docker container
  - For more details, see https://github.com/microsoft/rest-api-fuzz-testing

CI/CD with RAFT



Spin up containers

- Job-specified tools
- Service Under Test

Run tests

Report results

- Scalable parallel testing
- Add support for new tools without writing code
- Secret management
- VNET support
- Runs locally or in the cloud
- ode

RESTler Challenges/future work

- Better coverage "out of the box"
  - Improve mining of valid values from examples
  - Search for valid data payloads
  - Infer more dynamic objects
- Support continuous testing scenarios
  - CI/CD
  - Regression fuzzing
- Work with community on new security checkers
  - If you implement a new checker and would like to integrate or share your results, talk to us on github!

Conclusion

- APIs are everywhere: rapidly growing attack surface

- Stateful REST API fuzzing needed for deeper REST API coverage

- Thoroughly fuzzing services with a large or complex API is a hard problem
  - Automated tools like RESTler can help

- Please try RESTler and RAFT and give us your feedback!
  - https://github.com/microsoft/restler-fuzzer
  - https://github.com/microsoft/rest-api-fuzz-testing

Thank You!

Appendix

Fuzzing the API of real-world cloud service

- Authentication
- Pre-provisioning
- Dependencies between several APIs
- Naming constraints
- Resource creation patterns
  - Async
  - “Expensive”
  - Rate limited
- Hidden dependencies (e.g. /api/A/start, …, /api/A/stop)
- Many others